Exploit Development

Stack based Buffer Overflows

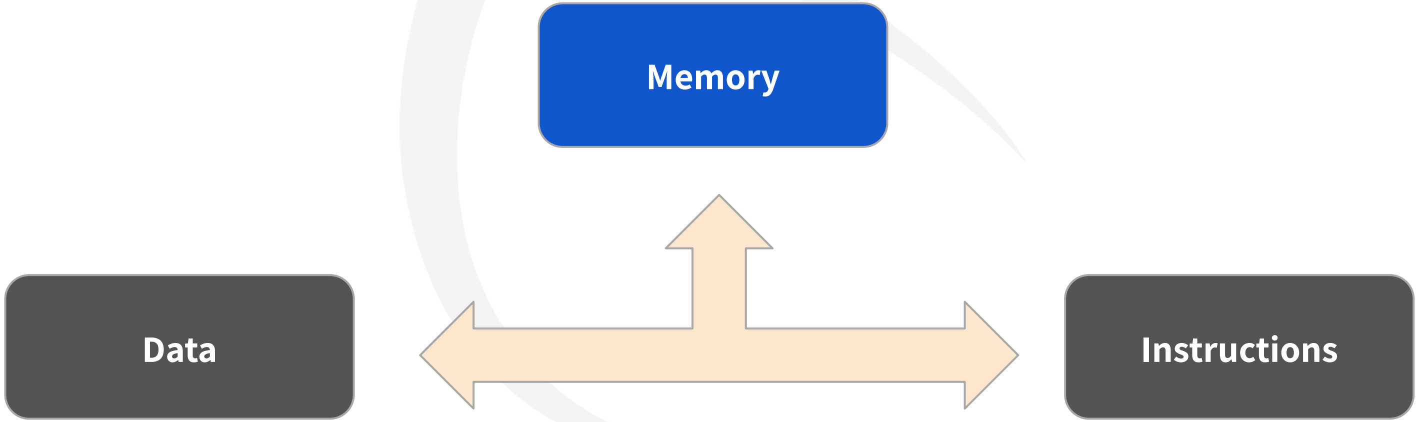
To Brag

- Adithyan AK - Head of OWASP Coimbatore
- 6+ Years into infosec
- Expertise in web app security, reverse engineering, exploit dev, malware analysis
- Author of several exploits & cves
- Speaker at various conferences, workshops (IITM Research Park, Defcon Trivandrum etc)
- Hall of fame in Microsoft, Apple, Intel, Avira, Oppo, etc
- Passion for making and breaking stuffs

Exploit Development - What & Why

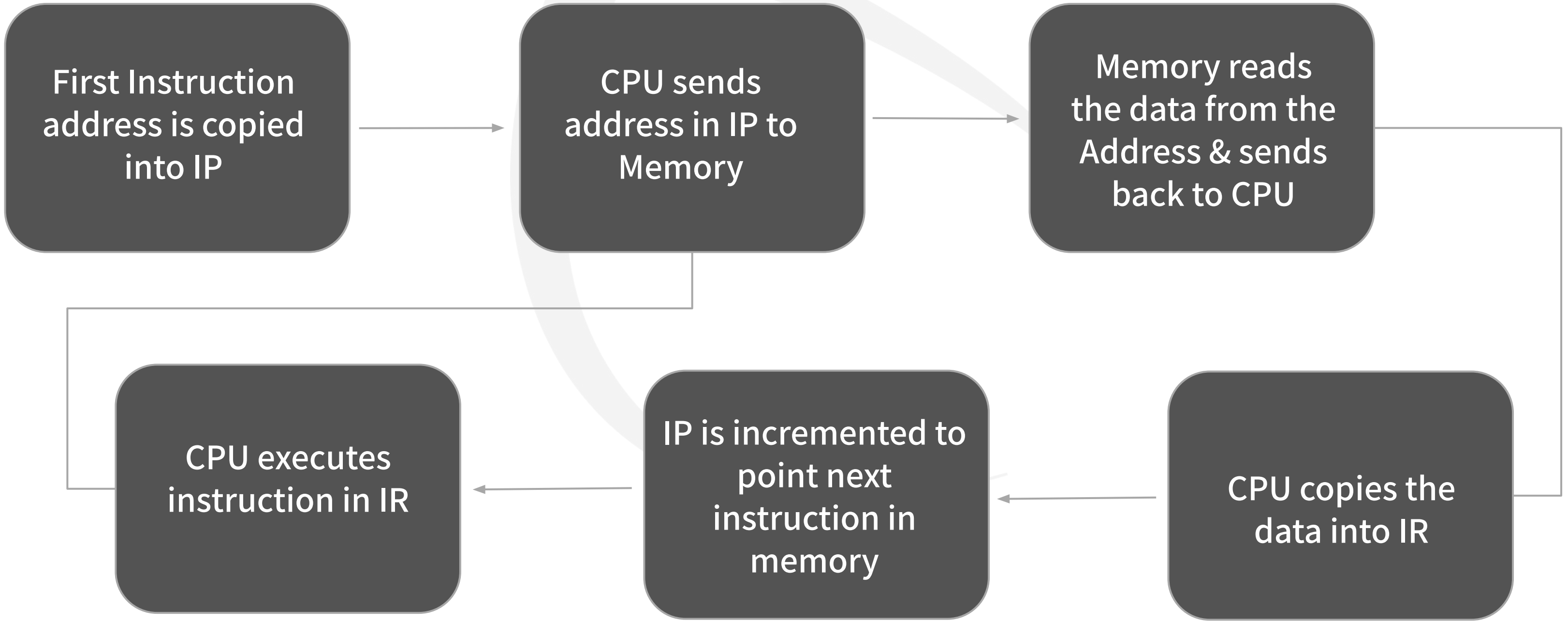
- Must have used dozens of exploits
- Download, Compile, Run -> B0000M!!!
- What if it’s a backdoor?
- Buffer Overflow
- Storage space
- Stack based -> local variables & return addresses
- Heap based -> dynamic data

Von Neumann Architecture



Program Execution in CPU

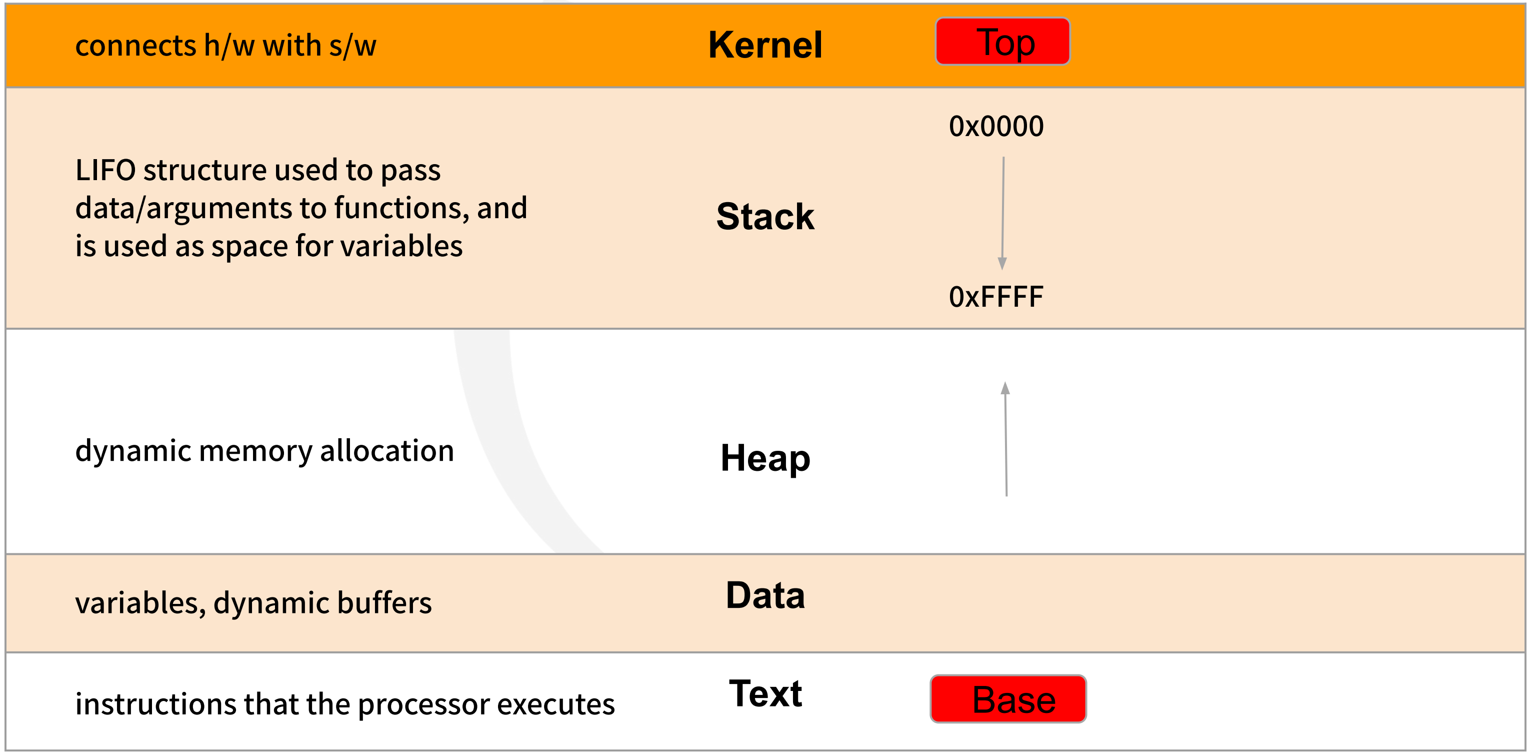
- Program -> Sequence of Instructions || IR -> Holds current Ins || IP -> Holds next instruction


First Instruction address is copied into IP
CPU sends address in IP to Memory
Memory reads the data from the Address & sends back to CPU
CPU copies the data into IR
IP is incremented to point next instruction in memory
CPU executes instruction in IR

CPU General Purpose Registers

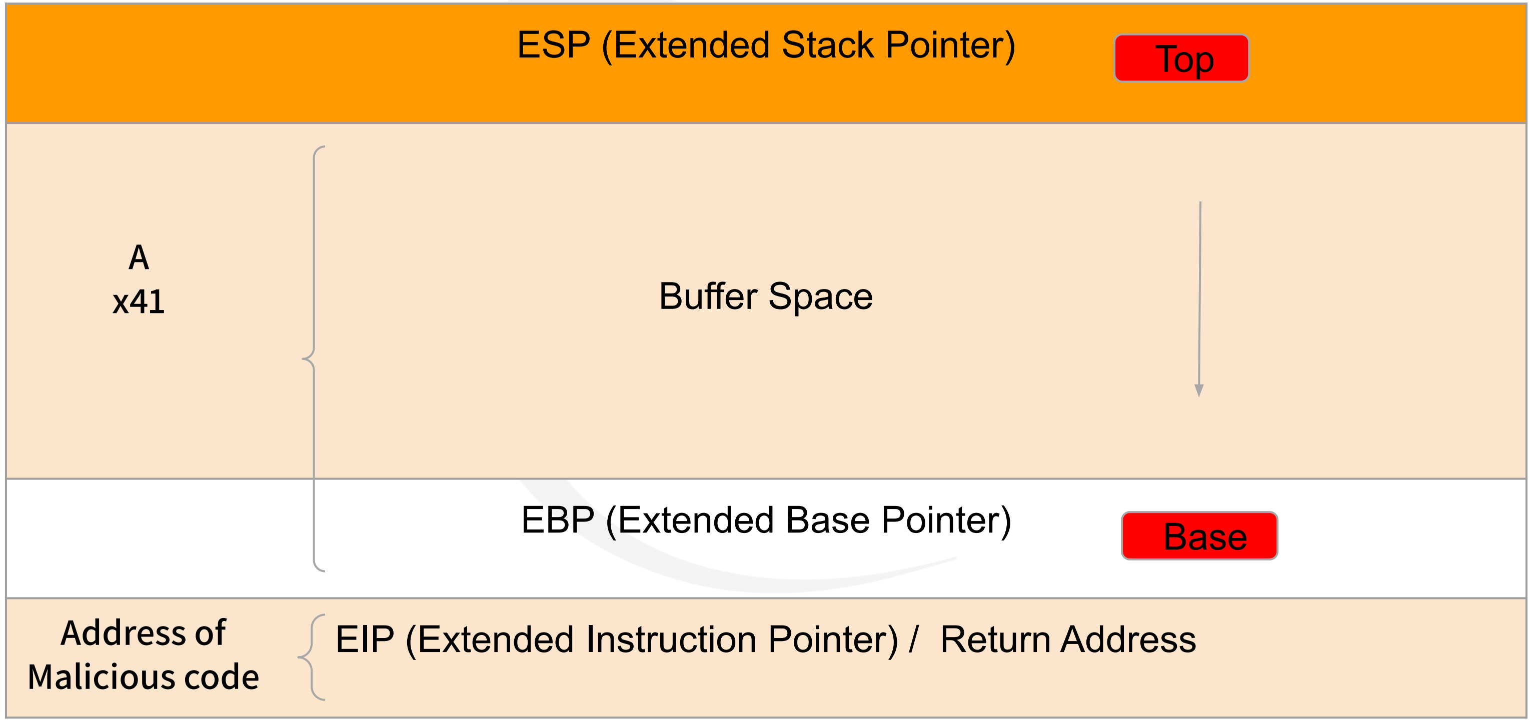
- EAX : accumulator : used for performing calculations, and used to store return values from function calls. Basic operations such as add, subtract, compare use this general-purpose register
- EBX : base (does not have anything to do with base pointer). It has no general purpose and can be used to store data.
- ECX : counter : used for iterations. ECX counts downward.
- EDX : data : extension of the EAX register. Allows for more complex calculations (multiply, divide)
- ESP : stack pointer
- EBP : base pointer
- ESI : source index : holds location of input data
- EDI : destination index : points to location of where result of data operation is stored
- EIP : instruction pointer

Anatomy of Program in Memory



connects h/w with s/w	**Kernel**	Top
LIFO structure used to pass data/arguments to functions, and is used as space for variables	**Stack**	0x0000 ↓ 0xFFFF
dynamic memory allocation	**Heap**	↑
variables, dynamic buffers	**Data**	
instructions that the processor executes	**Text**	Base

Anatomy of the Stack



ESP (Extended Stack Pointer)
Top
A
x41
Buffer Space
EBP (Extended Base Pointer)
Base
Address of
Malicious code
EIP (Extended Instruction Pointer) / Return Address

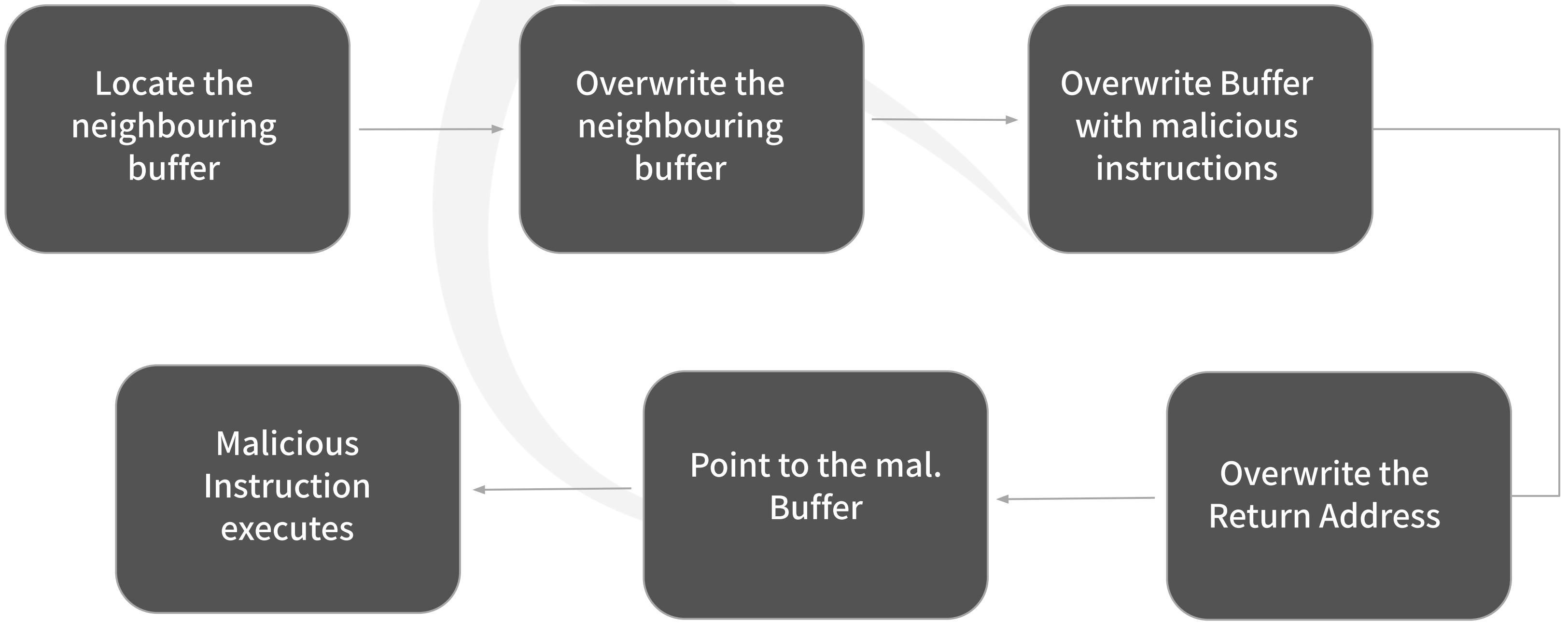
```c
int main(){
	char realPassword[20];
	char givenPassword[20];

	strncpy(realPassword, "ddddddddddddddd", 20);
	gets(givenPassword);

	if (0 == strncmp(givenPassword, realPassword, 20)){
		printf("SUCCESS!\n");
	}else{
		printf("FAILURE!\n");
	}
	printf("givenPassword: %s\n", givenPassword);
	printf("realPassword: %s\n", realPassword);
	return 0;
}
```

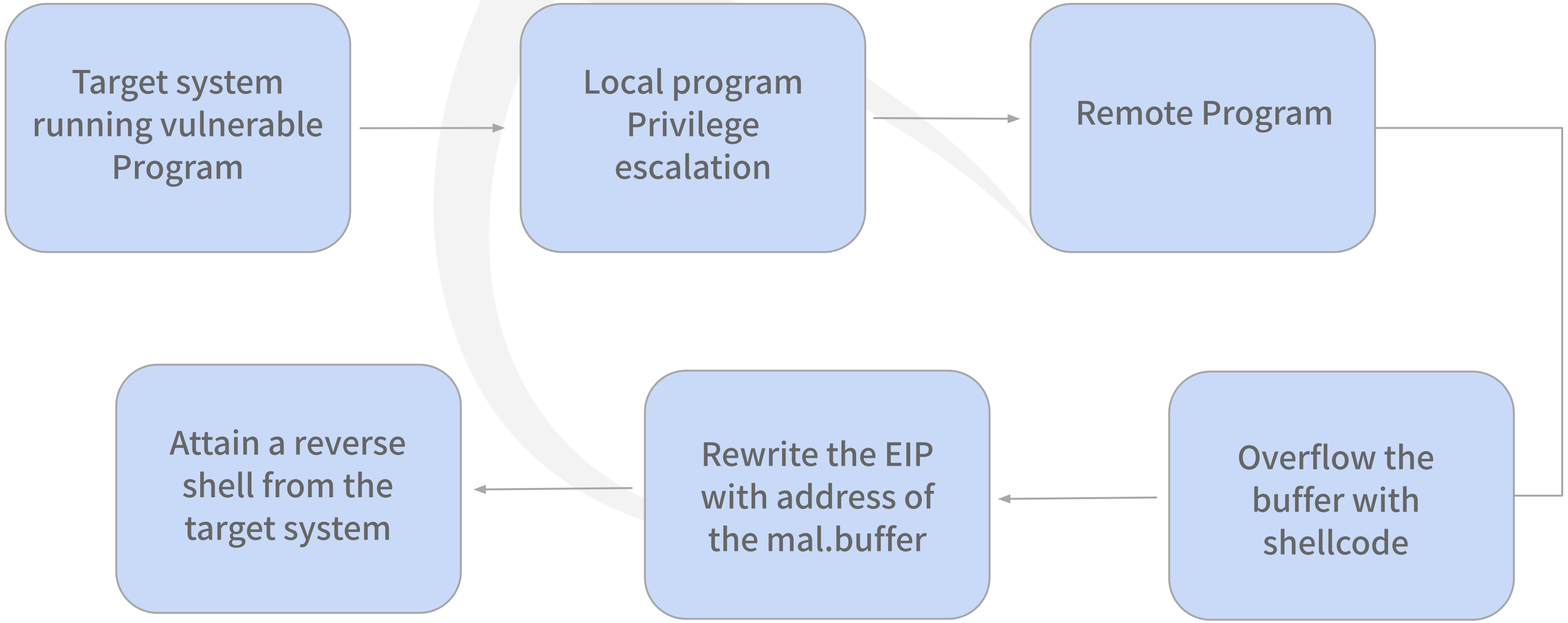
realPassword	givenPassword
ddddddddddd	input

Generic BOF Approach

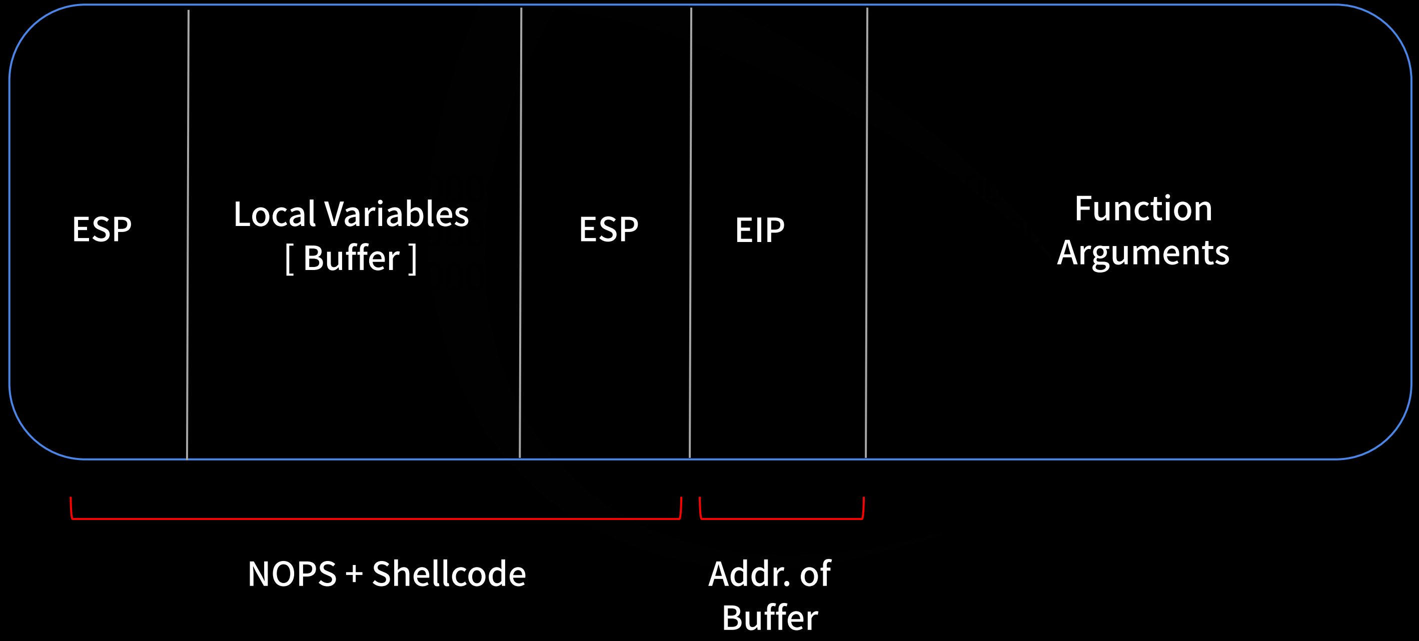


Locate the neighbouring buffer
Overwrite the neighbouring buffer
Overwrite Buffer with malicious instructions
Overwrite the Return Address
Point to the mal. Buffer
Malicious Instruction executes

Broad Overview of BOF Exploitation


Target system running vulnerable Program
Local program Privilege escalation
Remote Program
Overflow the buffer with shellcode
Rewrite the EIP with address of the mal.buffer
Attain a reverse shell from the target system

Stack Frame


ESP
Local Variables [Buffer]
ESP
EIP
Function Arguments
NOPS + Shellcode
Addr. of Buffer

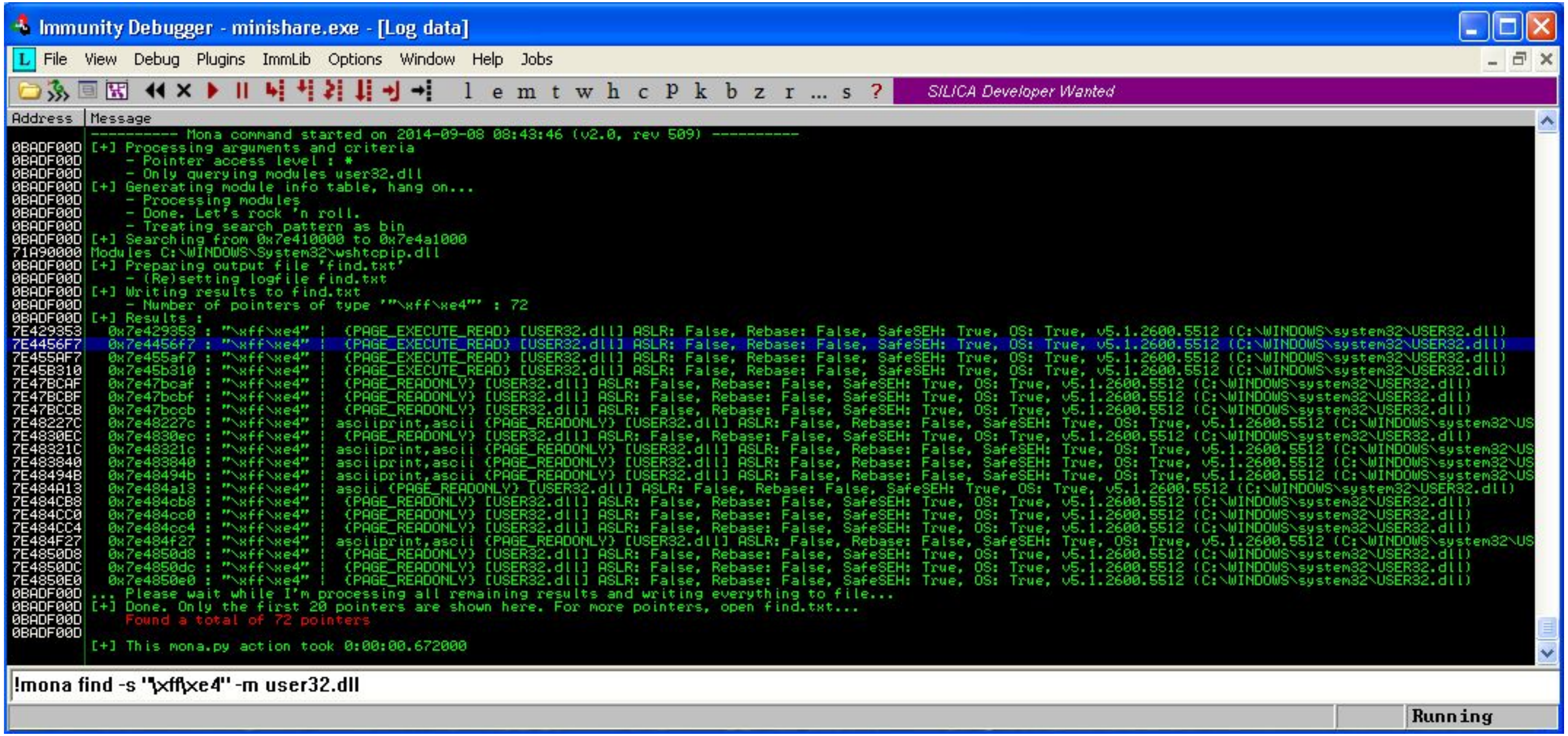
Fuzzing

- To identify the buffer length & capacity
- Stream of chars are sent
- Until the program breaks
- A = x41
- B = x42
- Find how many bytes break the buffer
- MSF Pattern create and offset
- Generate random string
- locate the position of the string reflected in EIP
- Overwrite EIP

Finding the Badchars

- unwanted characters that can break the shell codes.
- no universal set of bad characters
- different set of bad characters for every program
  - 00 for NULL
  - 0A for Line Feed \n
- Send the full list of the characters from 0x00 to 0xFF
- Check using debugger if input breaks
- If so, find the character that breaks it
- Remove the character from the list
- If input no longer breaks, use the rest of the characters to generate shellcode

Mona - by Corelan


Immunity Debugger - minishare.exe - [Log data]
L File View Debug Plugins ImmLib Options Window Help Jobs
l e m t w h c P k b z r ... s ?
SILICA Developer Wanted
Running

```
Address  Message
         ---------- Mona command started on 2014-09-08 08:43:46 (v2.0, rev 509) ----------
0BADF00D [+] Processing arguments and criteria
0BADF00D     - Pointer access level : *
0BADF00D     - Only querying modules user32.dll
0BADF00D [+] Generating module info table, hang on...
0BADF00D     - Processing modules
0BADF00D     - Done. Let's rock 'n roll.
0BADF00D     - Treating search pattern as bin
0BADF00D [+] Searching from 0x7e410000 to 0x7e4a1000
71A90000 Modules C:\WINDOWS\System32\wshtcpip.dll
0BADF00D [+] Preparing output file 'find.txt'
0BADF00D     - (Re)setting logfile find.txt
0BADF00D [+] Writing results to find.txt
0BADF00D     - Number of pointers of type '"\xff\xe4"' : 72
0BADF00D [+] Results :
7E429353   0x7e429353 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E4456F7   0x7e4456f7 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E455AF7   0x7e455af7 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E45B310   0x7e45b310 : "\xff\xe4" |  {PAGE_EXECUTE_READ} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E47BCAF   0x7e47bcaf : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E47BCBF   0x7e47bcbf : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E47BCCB   0x7e47bccb : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E48227C   0x7e48227c : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\US
7E4830EC   0x7e4830ec : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E48321C   0x7e48321c : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\US
7E483840   0x7e483840 : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\US
7E48494B   0x7e48494b : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\US
7E484A13   0x7e484a13 : "\xff\xe4" | ascii {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E484CB8   0x7e484cb8 : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E484CC0   0x7e484cc0 : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E484CC4   0x7e484cc4 : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E484F27   0x7e484f27 : "\xff\xe4" | asciiprint,ascii {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\US
7E4850D8   0x7e4850d8 : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E4850DC   0x7e4850dc : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
7E4850E0   0x7e4850e0 : "\xff\xe4" |  {PAGE_READONLY} [USER32.dll] ASLR: False, Rebase: False, SafeSEH: True, OS: True, v5.1.2600.5512 (C:\WINDOWS\system32\USER32.dll)
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open find.txt...
0BADF00D     Found a total of 72 pointers
0BADF00D
         [+] This mona.py action took 0:00:00.672000
```

```
!mona find -s "\xff\xe4" -m user32.dll
```

Generate Shellcode & PWN

Contact

adithyan-ak

adithyan_ak

akinfosec

akinfosec